Overlay Networks, Decentralized Systems and their Application

Exercise 02

University of Zurich

Reto WETTSTEIN
12-716-221
reto.wettstein2@uzh.ch

Solved together with Christian TRESCH 06-923-627

March 3, 2015

1 P2P Distributed Hash Tables

1 **Answer with true or false:**

1 **A central point will be always required with distributed indexing?**
False

2 **A distributed hash table requires that each node has a unique identifier?**
True

3 **P2P applications generate low amounts of traffic (compared to client-server) since that is the main purpose of having an P2P overlay?**
False

4 **An overlay on top of another overlay is possible**
True

2 **Cite and explain six key differences comparing the three strategies discussed in the lecture slides to store and retrieve data in P2P systems (Central server, Flooding search, Distributed indexing).**

CENTRAL SERVER: fuzzy queries, no false negatives, no robustness, not scalable, simple, fast

FLOODING SEARCH: fuzzy queries, false negatives, robustness, not scalable, simple, not fast

DISTRIBUTED INDEXING: no fuzzy queries, no false negatives, robustness, scalable, complex, quite fast

3 **What is the complexity of each lookup strategy listed below? And give a brief explanation of the reasons. Give the complexity in big-O notation, e.g. O(N^2).**

CENTRAL SERVER: O(1) $\rightarrow$ just ask the server.

FLOODING SEARCH: O(N) $\rightarrow$ ask every node if it has the data.

DISTRIBUTED INDEXING: O(log N) $\rightarrow$ any one node needs to coordinate with only a few other nodes in the system because each node is only responsible for a part of the address space.

4 **How is addressing handled in DHTs? How are identifiers chosen (for nodes and content)?**
Addressing in the DHT is the mapping of the content and the nodes into a linear address space with a hash function. This includes the association of parts of the address space to the DHT nodes (each node is only responsible for a part of the value range). This association of the data to nodes may change since nodes may disappear and reappear. A redundant storing is achieved with overlapping of the address parts in different nodes. Real and logical topology are thereby uncorrelated. Identifiers for nodes are random numbers, based on the public key or based on the IP. Identifiers for content can be calculated with hash(x) mod n.

5 **What happens if one node that is responsible for a given DHT address space fails? Explain two possible techniques to overcome this problem.**
If a node fails, the content corresponding to its address space will no longer be accessible and all routes going through this node will be interrupted. Possible solutions are:

POSSIBILITY 1: the use of redundant key/value pairs (data stored in more than one node) or

POSSIBILITY 2: the use of redundant/alternative routing paths

6 **Compare iterative routing with recursive routing. Give at least one advantage and one disadvantage for each technique.**

RECURSIVE ROUTING: The query is forwarded from one node to the next closest neighbour nodes until the searched node is found. This node replies to the originator.
+ Online status update
- No tracking of progress

ITERATIVE ROUTING: Each node replies to the originator of the query where to look next until the searched node is found.
+ Control
- Neighbour maintenance

7 **Cite method names, parameters, and return values of the generic interface of distributed hash tables.**

PUT(KEY, VALUE), NO RETURN: Called to store (redundantly) data with the specified key in the DHT.

GET(KEY), RETURNS VALUE: Called to get the data from the corresponding key. It is possible to get more than one result.

8 **Considering Kademlia/TomP2P, answer the following:**

1 **How many IDs are possible?**
A node ID has a length of 160 bit $\rightarrow 2^{160} \approx 1.5 \text{ x } 10^{48}$ different ID's are possible.

2 **Where is a key located?**
A key is located on the node whose ID is closest to the key.

3 **What is the XOR distance between 3 and 4?**
3 $\rightarrow$ 011 XOR 100 = 111, 3 bits to represent the XOR.

2 Challenge Task Preparation

This is my code for the programming task without import statements and javadoc. The output looks like this:

```
Peer with id 3 stored [Key: 12345, Value: Max Power]

Peer with id 5 received for key 12345 the data: Max Power

The peers with the following id's replied: 1, 2, 8
```

```java
package net.tomp2p.exercise;

public class Exercise2 {

   public static final int NUMBER_OF_PEERS = 10;
   public static final int STORING_PEER_INDEX = 2; // peerIndex is 1 smaller than peerId
   public static final int GETTER_PEER_INDEX = 4; // peerIndex is 1 smaller than peerId
   public static final Number160 KEY = new Number160(12345);
   public static final int PORT = 4001;


   public static void main(String[] args) {
      PeerDHT[] peers = null;

      try {
         peers = createAndAttachPeersDHT(NUMBER_OF_PEERS, PORT);
         bootstrap(peers);

         put(peers[STORING_PEER_INDEX], KEY, "Max Power");
         get(peers[GETTER_PEER_INDEX], KEY);

         peersShutdown(peers);
      } catch (IOException pEx) {
         pEx.printStackTrace();
      } catch (ClassNotFoundException pEx) {
         pEx.printStackTrace();
      }
   }


   public static PeerDHT[] createAndAttachPeersDHT(int nr, int port)
         throws IOException {
      PeerDHT[] peers = new PeerDHT[nr];
      for (int i = 0; i < nr; i++) {
         if (i == 0) {
            peers[0] = new PeerBuilderDHT(new PeerBuilder(new Number160(i +
                 1)).ports(port).start()).start();
         } else {
            peers[i] = new PeerBuilderDHT(new PeerBuilder(new Number160(i +
                 1)).masterPeer(peers[0].peer()).start()).start();
         }
      }

      return peers;
   }
```

```
    public static void bootstrap(PeerDHT[] peers) {
        // make perfect bootstrap, the regular can take a while
        for (int i = 0; i < peers.length; i++) {
            for (int j = 0; j < peers.length; j++) {
                peers[i].peerBean().peerMap().peerFound(peers[j].peerAddress(), null, null,
                    null);
            }
        }
    }


    public static void put(PeerDHT pPeer, Number160 pKey, String pValue)
            throws IOException {
        FuturePut futurePut = pPeer.put(pKey).data(new Data(pValue)).start();
        futurePut.awaitUninterruptibly();

        System.out.println("Peer with id " + pPeer.peerAddress().peerId().intValue() + "
             stored " + "[Key: " + pKey.intValue() + ", Value: " + pValue + "]");
    }


    public static Object get(PeerDHT pPeer, Number160 pKey)
            throws ClassNotFoundException, IOException {
        Object returnValue;

        FutureGet futureGet = pPeer.get(pKey).start();
        futureGet.awaitUninterruptibly();

        Set<Entry<PeerAddress, Map<Number640, Data>>> replies =
             futureGet.rawData().entrySet();

        returnValue = futureGet.data().object();
        System.out.println("\nPeer with id " + pPeer.peerAddress().peerId().intValue() + "
             received for key " + pKey.intValue() + " the data: " + returnValue);

        System.out.println("\nThe peers with the following id's replied:");
        Iterator<Entry<PeerAddress, Map<Number640, Data>>> iter = replies.iterator();
        while (iter.hasNext()) {
            Entry<PeerAddress, Map<Number640, Data>> entry = iter.next();
            System.out.println(entry.getKey().peerId().intValue());
        }

        return returnValue;
    }


    public static void peersShutdown(PeerDHT[] pPeers) {
        for (int i = 0; i < pPeers.length; i++) {
            pPeers[i].shutdown();
        }
    }
}
```